Searching Algorithms

=====================================================================================

Interface used

**public** **interface** ICodingProblems {

/\*\* 1. Find the Missing Number \*\*/

**public** **int** missNumber(**int** a[]);

/\*\* 2. Search an element in a sorted and rotated array \*\*/

**public** **int** searchInSortedAndRotatedArr(**int** a[], **int** x);

/\*\* 3. Median of two sorted arrays O(log(n)). \*\*/

**public** **int** medianOfTwoSortedArr(**int** a[], **int** b[]);

/\*\* 4. Two elements whose sum is closest to zero \*\*/

**public** **int**[] sumCloseToZero(**int** a[]);

/\*\* 5. Find the smallest and second smallest element in an array \*\*/

/\* use sortig application of 2 min element use bubble sort application \*/

/\*\* 6. Maximum and minimum of an array using minimum number of comparisons \*\*/

**public** **int**[] maxMinInMinComparision(**int** a[]);

/\*\* 7. k largest(or smallest) elements in an array | added Min Heap method \*\*/

/\* Refer to @IOrderStatistics.java ...point @1. \*/

/\*\* 8. Ceiling in a sorted array \*\*/

**public** **int** ceillingInSortedArr(**int** a[], **int** x, **int** l, **int** h);

/\*\* 9. Count number of occurrences (or frequency) in a sorted array \*\*/

/\* Refer @IBinarySearchBased.java...point @4. sub program \*/

/\*\* 10. Find the repeating and the missing | Added 3 new methods \*\*/

**public** **int**[] missingAndRepeatingInNNatural(**int** a[]);

/\*\* 11. Find a Fixed Point in a given array \*\*/

**public** **int** fixedPoint(**int** a[], **int** l, **int** h);

/\*\* 12. max element in array which is first increasing and then decreasing \*\*/

**public** **int** healInArray(**int** a[], **int** l, **int** h);

/\*\* 13. Find a pair with the given difference \*\*/

**public** **int**[] pairHaveXDiff(**int** a[], **int** x);

/\*\* 14. Find the k most frequent words from a file \*\*/

**public** List<FreqWord> kMostFrequent(String str[], **int** k);

/\*\* 15. Median of two sorted arrays of different sizes \*\*/

/\* @See IBinarySearchBased.java @ 2...program \*/

/\*\* 16. Find a peak element \*\*/

**public** **int** pickElement(**int** a[], **int** l, **int** h, **int** n);

/\*\*

\* 17. Given an array of of size n and a number k, find all elements that appear

\* more than n/k times

\*\*/

**public** List<Integer> elemMoreThanKTimes(**int** a[], **int** k);

/\*\* 18. Find the minimum element in a sorted and rotated array \*\*/

/\*

\* @see IArrayRotation.java...@ 5....a[findPivot(...)+1] ie next to poivot index

\*/

/\*\*

\* 19. Kth smallest element in a row-wise and column-wise sorted 2D array | Set

\* 1

\*/

/\* @see @IOrderStatistics.java ...@2. program \*/

/\*\* 20. Find k closest elements to a given value sorted arr \*\*/

**public** **int**[] kClogestElemInSotedArr(**int** a[], **int** val, **int** k);

/\*\* 21. Search in an almost sorted array \*\*/

**public** **int** searchInAlmostSotedArr(**int** a[], **int** x);

/\*\* 22. A Problem in Many Binary Search Implementations \*\*/

/\* Theory part \*/

/\*\* 23. Find the first repeating element in an array of integers \*\*/

**public** **int** firstRepetingElem(**int** a[]);

/\*\* 24. Find common elements in three sorted arrays \*\*/

**public** List<Integer> commonIn3Arrs(**int** a[], **int** b[], **int** c[]);

/\*\* 25. Count 1’s in a sorted binary array \*\*/

**public** **int** countOneInBinaryArr(**int** a[], **int** l, **int** h);

/\*\*

\* 26. Given a sorted array and a number x, find the pair in array whose sum is

\* closest to x

\*\*/

**public** List<Integer> sumCLogestToX(**int** a[], **int** x);

/\*\* 27. Find the closest pair from two sorted arrays \*\*/

**public** List<Integer> sumCLogestToX(**int** a[], **int** b[], **int** x);

/\*\* 28. K’th Smallest/Largest Element in Unsorted Array | Set 1 \*\*/

/\*\*

\* 29. K’th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear

\* Time)

\*\*/

/\*\*

\* 30. K’th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case

\* Linear Time)

\*\*/

/\* all above are common for minimum complxity O(k\*n) \*/

/\* @see @IOrderStatistics.java.. 1....program \*/

/\*\* 31. Find position of an element in a sorted array of infinite numbers \*\*/

// @Amazn

**public** **int** posInInfiniteLenArr(**int** a[], **int** x, **int** siseCheck, **int** sizeLimit);

/\*\*

\* 32. Given a sorted and rotated array, find if there is a pair with a given

\* sum

\*\*/

/\* in single array \*/

**public** List<Integer> sumEqualToX(**int** a[], **int** x);

/\* it two array \*/

**public** List<Integer> sumEqualToX(**int** a[], **int** b[], **int** x);

/\*\* 33. Find the largest pair sum in an unsorted array \*\*/

**public** List<Integer> largestPair(**int** a[]);

/\*\* 34. Find the nearest smaller numbers on left side in an array \*\*/

**public** **int**[] nearestSmallrOnLeftSideArr(**int** a[]);

/\*\* 35. K’th largest element in a stream \*\*/

/\* @see @IOrderStatistics.java....3....program8 \*/

/\*\* 36. Find a pair with maximum product in array of Integers \*\*/

/\* this is same max sum pair just use the product without absloute \*/

/\*\* 37. Find the element that appears once in a sorted array O(logn) \*\*/

**public** **int** uniqueElement(**int** a[], **int** l, **int** h);

/\*\* 38. Find the odd appearing element in O(Log n) time \*\*/

/\* logn in unsorted array \*/

**public** **int** searchIndex(**int** a[], **int** l, **int** h, **int** x);

**public** **int** oddAppearing(**int** a[]);

/\*\* 39. Find the largest three elements in an array O(n) \*\*/

**public** **int**[] largestThreeElement(**int** a[]);

}

Implementation of Interface

=====================================================================================

**public** **class** CodingProblemsImpl **implements** ICodingProblems {

/\*\* 1. Find the Missing Number \*\*/

**public** **int** missNumber(**int** a[]) {

**int** n = a.length;

**int** sum = 0;

**for** (**int** i = 0; i < a.length; i++)

sum += a[i];

**int** realNoSum = n \* (n + 1) / 2;

**return** realNoSum - sum;

}

/\*\* 2. Search an element in a sorted and rotated array \*\*/

**public** **int** searchInSortedAndRotatedArr(**int** a[], **int** x) {

IArrayRotation iar = **new** ArrayRotationImpl();

**return** iar.searchInSortedRotatedArr(a, x);

}

/\*\* 3. Median of two sorted arrays O(log(n)). \*\*/

**public** **int** medianOfTwoSortedArr(**int** a[], **int** b[]) {

IBinarySearchBased ibsb = **new** BinarySearchBasedImpl();

**return** ibsb.medianOfTwoSottedArr(a, b);

}

/\*\* 4. Two elements whose sum is closest to zero \*\*/

**public** **int**[] sumCloseToZero(**int** a[]) {

// combination of max,min +ve/-ve

**int** rs[] = **new** **int**[2];

Arrays.*sort*(a);

**if** (a[0] > 0) // ie all no are +ve

{

rs[0] = a[0];

rs[1] = a[1];

**return** rs;

}

**int** i = 0;

**int** j = a.length;

**while** (i < j && a[i] < 0)

i++;

// min -ve and max +ve combination

**int** min = Math.*abs*(a[0] + a[a.length - 1]);

rs[0] = a[0];

rs[1] = a[a.length - 1];

// max -ve and min +ve combination

**if** (Math.*abs*(a[i - 1] + a[i]) < min) {

min = a[i - 1] + a[i];

rs[0] = a[i - 1];

rs[1] = a[i];

}

// min -ve and min +ve combination

**if** (Math.*abs*(a[0] + a[i]) < min) {

rs[0] = a[0];

rs[1] = a[i];

}

// max -ve and max +ve combination

**if** (Math.*abs*(a[i - 1] + a[a.length - 1]) < min) {

rs[0] = a[i - 1];

rs[1] = a[a.length - 1];

}

**return** rs;

}

/\*\* 6. Maximum and minimum of an array using minimum number of comparisons \*\*/

// O(2n-1)

**public** **int**[] maxMinInMinComparision(**int** a[]) {

**int** n = a.length;

**int** rs[] = **new** **int**[2];

**int** temp;

**if** (a.length == 1) {

rs[0] = rs[1] = a[0];

**return** rs;

}

**for** (**int** i = 0; i < 2; i++) {

**for** (**int** j = i + 1; j < n; j++) {

**if** (i == 0) {

**if** (a[i] > a[j]) {

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

} **else** {

**if** (a[i] < a[j]) {

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

}

}

rs[0] = a[0];

rs[1] = a[1];

**return** rs;

}

/\*\* 8. Ceiling in a sorted array \*\*/

@Override

**public** **int** ceillingInSortedArr(**int** a[], **int** x, **int** l, **int** h) {

// first call need to check

**if** (l == 0 && h == a.length - 1) {

**if** (x < a[0])

**return** a[0];

**if** (x > a[h])

**return** -1;

}

**if** (l <= h) {

**int** mid = l + (h - l) / 2;

**if** (x == a[mid])

**return** a[mid];

**else** **if** (x <= a[mid] && x > a[mid - 1])

**return** a[mid];

**else** **if** (x > a[mid])

**return** ceillingInSortedArr(a, x, mid + 1, h);

**else**

**return** ceillingInSortedArr(a, x, l, mid - 1);

}

**return** -1;

}

/\*\* 9. Count number of occurrences (or frequency) in a sorted array \*\*/

/\* Refer @IBinarySearchBased.java...point @4. sub program \*/

/\*\* 10. Find the repeating and the missing | Added 3 new methods \*\*/

**public** **int**[] missingAndRepeatingInNNatural(**int** a[]) {

**int** n = a.length;

**int** rs[] = **new** **int**[2];

**int** sum = 0;

**int** x = 0;// misNo;

**int** y = 0;// repNo;

Set<Integer> set = **new** HashSet<>();

**for** (**int** i = 0; i < n; i++) {

sum += a[i];

**if** (set.contains(a[i]))

y = a[i];

**else**

set.add(a[i]);

}

x = Math.*abs*(n \* (n + 1) / 2 - sum + y);

rs[0] = y;

rs[1] = x;

**return** rs;

}

/\*\* 11. Find a Fixed Point in a given array(sorted) \*\*/

@Override

**public** **int** fixedPoint(**int** a[], **int** l, **int** h) {

**if** (l <= h) {

**int** mid = l + (h - l) / 2;

**if** (mid == a[mid])

**return** mid;

**else** **if** (mid > a[mid])

**return** fixedPoint(a, mid + 1, h);

**else**

**return** fixedPoint(a, l, mid - 1);

}

**return** -1;

}

/\*\* 12. max element in array which is first increasing and then decreasing \*\*/

// this is nothing but find the max element in array O(log(n))

@Override

**public** **int** healInArray(**int** arr[], **int** low, **int** high) {

/\* Base Case: Only one element is present in arr[low..high] \*/

**if** (low == high)

**return** arr[low];

/\*

\* If there are two elements and first is greater then the first element is

\* maximum

\*/

**if** ((high == low + 1) && arr[low] >= arr[high])

**return** arr[low];

/\*

\* If there are two elements and second is greater then the second element is

\* maximum

\*/

**if** ((high == low + 1) && arr[low] < arr[high])

**return** arr[high];

**int** mid = (low + high) / 2; /\* low + (high - low)/2; \*/

/\*

\* If we reach a point where arr[mid] is greater than both of its adjacent

\* elements arr[mid-1] and arr[mid+1], then arr[mid] is the maximum element

\*/

**if** (arr[mid] > arr[mid + 1] && arr[mid] > arr[mid - 1])

**return** arr[mid];

/\*

\* If arr[mid] is greater than the next element and smaller than the previous

\* element then maximum lies on left side of mid

\*/

**if** (arr[mid] > arr[mid + 1] && arr[mid] < arr[mid - 1])

**return** healInArray(arr, low, mid - 1);

**else** // when arr[mid] is greater than arr[mid-1] and smaller than arr[mid+1]

**return** healInArray(arr, mid + 1, high);

}

/\*\* 13. Find a pair with the given difference \*\*/

@Override

**public** **int**[] pairHaveXDiff(**int** a[], **int** x) {

**int** rs[] = **new** **int**[2];

Set<Integer> set = **new** HashSet<Integer>();

**for** (**int** i = 0; i < a.length; i++) {

**if** (set.contains(a[i])) {

rs[0] = a[i] - x;

rs[1] = a[i];

**return** rs;

} **else** {

set.add(a[i] + x);

}

}

**return** rs;

}

/\*\* 14. Find the k most frequent words from a file \*\*/

@Override

**public** List<FreqWord> kMostFrequent(String str[], **int** k) {

List<FreqWord> rsList = **new** ArrayList<>();

Map<String, Integer> map = **new** HashMap<>();

**for** (**int** i = 0; i < str.length; i++) {

**if** (map.containsKey(str[i])) {

map.put(str[i], map.get(str[i]) + 1);

} **else**

map.put(str[i], 1);

}

List<FreqWord> words = **new** ArrayList<>();

**for** (Map.Entry<String, Integer> entry : map.entrySet()) {

words.add(**new** FreqWord(entry.getKey(), entry.getValue()));

}

Collections.*sort*(words);

**for** (**int** i = 0; i < k; i++)

rsList.add(words.get(i));

**return** rsList;

}

/\*\* 15. Median of two sorted arrays of different sizes \*\*/

/\* @See IBinarySearchBased.java @ 2...program \*/

/\*\* 16. Find a peak element \*\*/

@Override

**public** **int** pickElement(**int** a[], **int** l, **int** h, **int** n) {

**int** mid = (l + h) / 2;

**if** ((mid == 0 || a[mid] >= a[mid - 1]) && (mid == n - 1 || a[mid] >= a[mid + 1]))

**return** a[mid];

**else** **if** (mid > 0 && a[mid] < a[mid - 1])

**return** pickElement(a, l, mid - 1, n);

**else**

**return** pickElement(a, mid + 1, h, n);

}

/\*\*

\* 17. Given an array of of size n and a number k, find all elements that appear

\* more than n/k times

\*\*/

@Override

**public** List<Integer> elemMoreThanKTimes(**int** a[], **int** k) {

List<Integer> rsList = **new** ArrayList<>();

Map<Integer, Integer> map = **new** HashMap<>();

**for** (**int** i = 0; i < a.length; i++) {

**if** (map.containsKey(a[i])) {

map.put(a[i], map.get(a[i]) + 1);

} **else**

map.put(a[i], 1);

}

**for** (Map.Entry<Integer, Integer> entry : map.entrySet()) {

**if** (entry.getValue() > a.length / k)

rsList.add(entry.getKey());

}

**return** rsList;

}

/\*\* 18. Find the minimum element in a sorted and rotated array \*\*/

/\*

\* @see IArrayRotation.java...@ 5....a[findPivot(...)+1] ie next to poivot index

\*/

/\*\*

\* 19. Kth smallest element in a row-wise and column-wise sorted 2D array | Set

\* 1

\*/

/\* @see @IOrderStatistics.java ...@2. program \*/

/\*\* 20. Find k closest elements to a given value sorted arr \*\*/

@Override

**public** **int**[] kClogestElemInSotedArr(**int** a[], **int** val, **int** k) {

**int** n = a.length;

**int** rs[] = **new** **int**[k];

**int** i, j, l;

l = 0;

// O(log(n))

// first find the val in arry if not found then fid the clogest one

**int** ind = AlgoUtils.*binarySearch*(a, 0, n - 1, val);

**if** (ind == -1) {

ind = AlgoUtils.*findCrossOver*(a, 0, n - 1, val);

rs[k++] = a[ind];

l++;

}

// k+logn

i = ind - 1;

j = ind + 1;

**while** (l < k && j < a.length && i > 0) {

**if** (val - a[i] > a[j] - val) {

rs[l++] = a[j];

j++;

} **else** {

rs[l++] = a[i];

i--;

}

}

// if one end is over

**if** (i == 0) {

**while** (l < k && j < a.length)

rs[l++] = a[j++];

}

**if** (j == n) {

**while** (i > 0 && l < k)

rs[l++] = a[i--];

}

**return** rs;

}

/\*\* 21. Search in an almost sorted array \*\*/

**public** **int** searchInAlmostSotedArr(**int** a[], **int** x) {

**return** **new** BinarySearchBasedImpl().binarySearchInAllmostSoted(a, 0, a.length - 1, x);

}

/\*\* 23. Find the first repeating element in an array of integers \*\*/

@Override

**public** **int** firstRepetingElem(**int** a[]) {

Map<Integer, Integer> map = **new** HashMap<>();

**int** minInd = Integer.***MAX\_VALUE***;

**for** (**int** i = 0; i < a.length; i++) {

**if** (map.containsKey(a[i])) {

{

**if** (map.get(a[i]) < minInd)

minInd = map.get(a[i]);

}

} **else**

map.put(a[i], i);

}

**return** minInd != Integer.***MAX\_VALUE*** ? a[minInd] : minInd;

}

/\*\* 24. Find common elements in three sorted arrays \*\*/

@Override

**public** List<Integer> commonIn3Arrs(**int** a[], **int** b[], **int** c[]) {

**int** n1 = a.length;

**int** n2 = b.length;

**int** n3 = c.length;

**int** i, j, k;

i = j = k = 0;

// O(n1+n2+n3)

List<Integer> list = **new** ArrayList<>();

**while** (i < n1 && j < n2 && k < n3) {

**if** (a[i] == b[j] && b[j] == c[k]) {

list.add(a[i]);

i++;

j++;

k++;

} **else** **if** (a[i] < b[j])

i++;

**else** **if** (b[j] < c[k])

j++;

**else**

k++;

}

**return** list;

}

/\*\* 25. Count 1’s in a sorted binary array \*\*/

@Override // sorted in non-increasing order

**public** **int** countOneInBinaryArr(**int** a[], **int** l, **int** h) {

**if** (l <= h) {

// get the middle index

**int** mid = l + (h - l) / 2;

// check if the element at middle index is last 1

**if** ((mid == h || a[mid + 1] == 0) && (a[mid] == 1))

**return** mid + 1;

// If element is not last 1, recur for right side

**if** (a[mid] == 1)

**return** countOneInBinaryArr(a, (mid + 1), h);

// else recur for left side

**return** countOneInBinaryArr(a, l, (mid - 1));

}

**return** 0;

}

/\*\*

\* 26. Given a sorted array and a number x, find the pair in array whose sum is

\* closest to x

\*\*/

@Override

**public** List<Integer> sumCLogestToX(**int** a[], **int** x) {

List<Integer> list = **new** ArrayList<>();

**int** ind1, ind2;

ind1 = ind2 = 0;

**int** i = 0;

**int** j = a.length - 1;

**int** diff = Integer.***MAX\_VALUE***;

**while** (i < j) {

**if** (Math.*abs*(a[i] + a[j] - x) < diff) {

ind1 = i;

ind2 = j;

diff = Math.*abs*(a[i] + a[j] - x);

i++;

j--;

}

// If this pair has more sum, move to smaller values.

**else** **if** (a[i] + a[j] > x)

j--;

**else** // Move to larger values

i++;

}

list.add(a[ind1]);

list.add(a[ind2]);

**return** list;

}

/\*\* 27. Find the closest pair from two sorted arrays \*\*/

@Override

**public** List<Integer> sumCLogestToX(**int** a[], **int** b[], **int** x) {

**int** n1 = a.length;

**int** j = b.length - 1;

**int** i = 0;

**int** diff = Integer.***MAX\_VALUE***;

**int** ind1 = 0;

**int** ind2 = 0;

// O(n1+n2+n3)

List<Integer> list = **new** ArrayList<>();

**while** (i < n1 && j >= 0) {

**if** (Math.*abs*(a[i] + b[j] - x) < diff) {

diff = Math.*abs*(a[i] + b[j] - x);

ind1 = i;

ind2 = j;

i++;

j--;

} **else** **if** (a[i] + a[j] > x)

j--;

**else**

i++;

}

list.add(a[ind1]);

list.add(b[ind2]);

**return** list;

}

/\*\* 28. K’th Smallest/Largest Element in Unsorted Array | Set 1 \*\*/

/\*\*

\* 29. K’th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear

\* Time)

\*\*/

/\*\*

\* 30. K’th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case

\* Linear Time)

\*\*/

/\* all above are common for minimum complxity O(k\*n) \*/

/\* @see @IOrderStatistics.java.. 1....program \*/

/\*\* 31. Find position of an element in a sorted array of infinite numbers \*\*/

@Override // @Amazn

**public** **int** posInInfiniteLenArr(**int** a[], **int** x, **int** siseCheck, **int** sizeLimit) {

// infinite menas length is unknown

// try for first for last index 1000,2000 etc

**int** l = 0;

**int** h = 0;

**int** i = 0;

**try** {

**for** (i = 0; i < sizeLimit; i += siseCheck) {

**if** (a[i] <= x && x <= a[i + siseCheck]) {

l = i;

h = i + siseCheck;

**return** AlgoUtils.*binarySearch*(a, l, h, x);

}

}

} **catch** (ArrayIndexOutOfBoundsException e) {

**return** posInInfiniteLenArr(a, x, siseCheck / 10, i + siseCheck);

}

**return** -1;

}

/\*\*

\* 32. Given a sorted and rotated array, find if there is a pair with a given

\* sum

\*\*/

/\* in single array \*/

@Override

**public** List<Integer> sumEqualToX(**int** a[], **int** x) {

**int** j = a.length - 1;

**int** i = 0;

**int** ind1 = 0;

**int** ind2 = 0;

// O(n1+n2+n3)

List<Integer> list = **new** ArrayList<>();

**while** (i < j) {

**if** (Math.*abs*(a[i] + a[j] - x) == 0) {

ind1 = i;

ind2 = j;

**break**;

} **else** **if** (a[i] + a[j] > x)

j--;

**else**

i++;

}

**if** (i != j) {// ie . if found

list.add(a[ind1]);

list.add(a[ind2]);

}

**return** list;

}

/\* in two array \*/

@Override

**public** List<Integer> sumEqualToX(**int** a[], **int** b[], **int** x) {

**int** n1 = a.length;

**int** j = b.length - 1;

**int** i = 0;

**int** ind1 = 0;

**int** ind2 = 0;

// O(n1+n2+n3)

List<Integer> list = **new** ArrayList<>();

**while** (i < n1 && j >= 0) {

**if** (Math.*abs*(a[i] + b[j] - x) == 0) {

ind1 = i;

ind2 = j;

**break**;

} **else** **if** (a[i] + b[j] > x)

j--;

**else**

i++;

}

**if** (i != n1 && j != -1) {// ie. if found

list.add(a[ind1]);

list.add(b[ind2]);

}

**return** list;

}

/\*\* 33. Find the largest pair sum in an unsorted array \*\*/

@Override

**public** List<Integer> largestPair(**int** a[]) {

List<Integer> list = **new** ArrayList<>();

**int** firstMax = 0;

**int** secondMax = 0;

**if** (a[0] > a[1]) {

firstMax = a[0];

secondMax = a[1];

} **else** {

firstMax = a[1];

secondMax = a[0];

}

**for** (**int** i = 2; i < a.length; i++) {

**if** (a[i] > firstMax) {

secondMax = firstMax;

firstMax = a[i];

}

}

list.add(firstMax);

list.add(secondMax);

**return** list;

}

/\*\* 34. Find the nearest smaller numbers on left side in an array \*\*/

@Override // O(n)

**public** **int**[] nearestSmallrOnLeftSideArr(**int** a[]) {

**int** n = a.length;

**int** b[] = **new** **int**[n];

Stack<Integer> stack = **new** Stack<>();

**for** (**int** i = 0; i < n; i++) {

**while** (!stack.isEmpty() && stack.peek() > a[i])

stack.pop();

**if** (!stack.isEmpty())

b[i] = stack.peek();

stack.push(a[i]);

}

**return** b;

}

/\*\* 35. K’th largest element in a stream \*\*/

/\* @see @IOrderStatistics.java....3....program8 \*/

/\*\* 36. Find a pair with maximum product in array of Integers \*\*/

/\* this is same max sum pair just use the product without absloute \*/

/\*\* 37. Find the element that appears once in a sorted array O(logn) \*\*/

@Override

**public** **int** uniqueElement(**int** a[], **int** l, **int** h) {

**if** (l <= h) {

**if** (h == l)

**return** a[h];

**int** mid = (l + h) / 2;

**int** x = -1;

**if** (mid % 2 == 0) {

**if** (a[mid] == a[mid + 1])

x = uniqueElement(a, mid + 2, h);

**if** (x != -1)

**return** x;

**else**

x = uniqueElement(a, l, mid);

**if** (x != -1)

**return** x;

} **else** {

**if** (a[mid] == a[mid - 1])

x = uniqueElement(a, mid + 1, h);

**if** (x != -1)

**return** x;

**else**

x = uniqueElement(a, l, mid - 1);

**if** (x != -1)

**return** x;

}

}

**return** -1;

}

/\*\* 38. Find the odd appearing element in O(Log n) time non sorted \*\*/

/\* logn in unsorted array **TODO** \*/

@Override // it take O(n)

**public** **int** searchIndex(**int** a[], **int** l, **int** h, **int** x) {

**if** (l <= h) {

**int** mid = (l + h) / 2;

**if** (a[mid] == x) {

**return** mid;

} **else** {

**return** Math.*max*(searchIndex(a, mid + 1, h, x), searchIndex(a, l, mid - 1, x));

}

}

**return** -1;

}

@Override

**public** **int** oddAppearing(**int** a[]) {

**return** uniqueElement(a, 0, a.length - 1);

}

/\*\* 39. Find the largest three elements in an array O(n) \*\*/

**public** **int**[] largestThreeElement(**int** a[]) {

**if** (a.length < 3)

**return** **null**;

**int** b[] = **new** **int**[3];

**int** i, first, second, third;

third = first = second = Integer.***MIN\_VALUE***;

**for** (i = 0; i < a.length; i++) {

/\* If current element is smaller than first \*/

**if** (a[i] > first) {

third = second;

second = first;

first = a[i];

}

/\* If arr[i] is in between first and second then update second \*/

**else** **if** (a[i] > second) {

third = second;

second = a[i];

}

**else** **if** (a[i] > third)

third = a[i];

}

b[0] = third;

b[1] = second;

b[2] = first;

**return** b;

}

}

Test cases

====================================================================

**public** **class** ICodingProblemsTest {

**public** ICodingProblems icp = **null**;

@Before

**public** **void** init() {

icp = **new** CodingProblemsImpl();

}

/\*\* 1. Find the Missing Number \*\*/

@Test

**public** **void** missNumberTest() {

**int** a[] = { 1, 2, 4, 6, 0, 3, 7, 8 };

Assert.*assertTrue*(icp.missNumber(a) == 5);

}

/\*\* 2. Search an element in a sorted and rotated array \*\*/

@Test

**public** **void** searchInSortedAndRotatedArrTest() {

**int** b[] = { 4, 5, 6, 7, 8, 9, 1, 2, 3 };

// System.out.println(iar.searchInSortedRotatedArr(b, 4));

Assert.*assertTrue*(icp.searchInSortedAndRotatedArr(b, 8) == 4);

Assert.*assertTrue*(icp.searchInSortedAndRotatedArr(b, 3) == 8);

Assert.*assertTrue*(icp.searchInSortedAndRotatedArr(b, 4) == 0);

}

/\*\* 3. Median of two sorted arrays O(log(n)). \*\*/

@Test

**public** **void** medianOfTwoSortedArrTest() {

**int** a[] = { 1, 3 };

**int** b[] = { 2, 4 };

Assert.*assertTrue*(icp.medianOfTwoSortedArr(a, b) == 2);

**int** ar1[] = { 1, 12, 15, 26, 38 };

**int** ar2[] = { 2, 13, 17, 30, 45 };

Assert.*assertTrue*(icp.medianOfTwoSortedArr(ar1, ar2) == 16);

}

/\*\* 4. Two elements whose sum is closest to zero \*\*/

@Test

**public** **void** sumCloseToZeroTest() {

**int** arr[] = { 1, 60, -10, 70, -80, 85 };

arr = icp.sumCloseToZero(arr);

Assert.*assertTrue*(arr[0] == -80);

Assert.*assertTrue*(arr[1] == 85);

**int** b[] = { -1, 5, -2, 8, -3, 9, -4 };

arr = icp.sumCloseToZero(b);

Assert.*assertTrue*(arr[0] == -4);

Assert.*assertTrue*(arr[1] == 5);

}

/\*\* 6. Maximum and minimum of an array using minimum number of comparisons \*\*/

@Test

**public** **void** maxMinInMinComparisionTest() {

**int** b[] = { -1, 5, -2, 8, -3, 9, -4 };

b = icp.maxMinInMinComparision(b);

Assert.*assertTrue*(b[0] == -4);

Assert.*assertTrue*(b[1] == 9);

}

/\*\* 8. Ceiling in a sorted array \*\*/

@Test

**public** **void** ceillingInSortedArrTest() {

**int** a[] = { 1, 2, 8, 10, 10, 12, 19 };

**int** n = a.length - 1;

Assert.*assertTrue*(icp.ceillingInSortedArr(a, 0, 0, n) == 1);

Assert.*assertTrue*(icp.ceillingInSortedArr(a, 20, 0, n) == -1);

Assert.*assertTrue*(icp.ceillingInSortedArr(a, 1, 0, n) == 1);

Assert.*assertTrue*(icp.ceillingInSortedArr(a, 5, 0, n) == 8);

}

/\*\* 9. Count number of occurrences (or frequency) in a sorted array \*\*/

/\* Refer @IBinarySearchBased.java...point @4. sub program \*/

/\*\* 10. Find the repeating and the missing | Added 3 new methods \*\*/

@Test

**public** **void** missingAndRepeatingInNNaturalTest() {

**int** arr[] = { 1, 3, 4, 5, 5, 6, 2 };

arr = icp.missingAndRepeatingInNNatural(arr);

Assert.*assertTrue*(arr[0] == 5);

Assert.*assertTrue*(arr[1] == 7);

}

/\*\* 11. Find a Fixed Point in a given array \*\*/

@Test

**public** **void** fixedPointTest() {

**int** arr[] = { -10, -5, 0, 3, 7 };

Assert.*assertTrue*(icp.fixedPoint(arr, 0, arr.length - 1) == 3);

}

/\*\* 12. max element in array which is first increasing and then decreasing \*\*/

@Test

**public** **void** healInArrayTest() {

**int** arr[] = { 8, 10, 20, 80, 100, 200, 400, 500, 3, 2, 1 };

Assert.*assertTrue*(icp.healInArray(arr, 0, arr.length - 1) == 500);

**int** arra[] = { 1, 3, 50, 10, 9, 7, 6 };

Assert.*assertTrue*(icp.healInArray(arra, 0, arra.length - 1) == 50);

**int** a[] = { 10, 20, 30, 40, 50 };

Assert.*assertTrue*(icp.healInArray(a, 0, a.length - 1) == 50);

**int** b[] = { 120, 100, 80, 20, 0 };

Assert.*assertTrue*(icp.healInArray(b, 0, b.length - 1) == 120);

}

/\*\* 13. Find a pair with the given difference \*\*/

@Test

**public** **void** pairHaveXDiffTest() {

**int** arr[] = { 1, 8, 30, 40, 100 };

arr = icp.pairHaveXDiff(arr, 39);

Assert.*assertTrue*(arr[0] == 1);

Assert.*assertTrue*(arr[1] == 40);

}

/\*\* 14. Find the k most frequent words from a file \*\*/

@Test

**public** **void** kMostFrequentTest() {

String str = "Welcome to the world of Geeks"

+ "This portal has been created to provide well written well thought and well explained "

+ "solutions for selected questions If you like Geeks for Geeks and would like to contribute "

+ "here is your chance You can write article and mail your article to contribute at "

+ "geeksforgeeks org See your article appearing on the Geeks for Geeks main page and help "

+ "thousands of other Geeks";

String tokns[] = str.split(" ");

List<FreqWord> rsList = icp.kMostFrequent(tokns, 5);

// System.out.println(Arrays.toString(rsList.toArray()));

// [(Geeks,5), (and,4), (to,4), (for,3), (your,3)]

Assert.*assertTrue*(rsList.get(0).word.equals("Geeks"));

Assert.*assertTrue*(rsList.get(0).frq == 5);

}

/\*\* 15. Median of two sorted arrays of different sizes \*\*/

/\* @See IBinarySearchBased.java @ 2...program \*/

/\*\* 16. Find a peak element \*\*/

@Test

**public** **void** pickElementTest() {

**int** arr[] = { 1, 3, 20, 4, 1, 0 };

Assert.*assertTrue*(icp.pickElement(arr, 0, arr.length - 1, arr.length) == 20);

}

/\*\*

\* 17. Given an array of of size n and a number k, find all elements that appear

\* more than n/k times

\*\*/

@Test

**public** **void** elemMoreThanKTimesTest() {

**int** a[] = { 3, 1, 2, 2, 1, 2, 3, 3 };

List<Integer> rsList = icp.elemMoreThanKTimes(a, 4);

Assert.*assertTrue*(rsList.size() == 2);

Assert.*assertTrue*(rsList.get(0) == 3 || rsList.get(0) == 2);

Assert.*assertTrue*(rsList.get(1) == 3 || rsList.get(1) == 2);

}

/\*\* 18. Find the minimum element in a sorted and rotated array \*\*/

/\*

\* @see IArrayRotation.java...@ 5....a[findPivot(...)+1] ie next to poivot index

\*/

/\*\*

\* 19. Kth smallest element in a row-wise and column-wise sorted 2D array | Set

\* 1

\*/

/\* @see @IOrderStatistics.java ...@2. program \*/

/\*\* 20. Find k closest elements to a given value sorted arr \*\*/

@Test

**public** **void** kClogestElemInSotedArrTest() {

**int** arr[] = { 12, 16, 22, 30, 35, 39, 42, 45, 48, 50, 53, 55, 56 };

**int** x = 35, k = 4;

arr = icp.kClogestElemInSotedArr(arr, x, k);

**int** rs[] = { 39, 30, 42, 45 };

**for** (**int** i = 0; i < rs.length; i++)

Assert.*assertTrue*(rs[i] == arr[i]);

}

/\*\* 21. Search in an almost sorted array \*\*/

@Test

**public** **void** searchInAlmostSotedArrTest() {

**int** a[] = { 3, 2, 10, 4, 40 };

Assert.*assertTrue*(icp.searchInAlmostSotedArr(a, 4) == 3);

}

/\*\* 23. Find the first repeating element in an array of integers \*\*/

@Test

**public** **void** firstRepetingElemTest() {

**int** arr[] = { 10, 5, 3, 4, 3, 5, 6 };

**int** brr[] = { 6, 10, 5, 4, 9, 120, 4, 6, 10 };

Assert.*assertTrue*(icp.firstRepetingElem(arr) == 5);

Assert.*assertTrue*(icp.firstRepetingElem(brr) == 6);

}

/\*\* 24. Find common elements in three sorted arrays \*\*/

@Test

**public** **void** commonIn3ArrsTest() {

**int** ar1[] = { 1, 5, 10, 20, 40, 80 };

**int** ar2[] = { 6, 7, 20, 80, 100 };

**int** ar3[] = { 3, 4, 15, 20, 30, 70, 80, 120 };

// Output: 20, 80

List<Integer> list = icp.commonIn3Arrs(ar1, ar2, ar3);

Assert.*assertTrue*(list.get(0) == 20);

Assert.*assertTrue*(list.get(1) == 80);

**int** ar11[] = { 1, 5, 5 };

**int** ar21[] = { 3, 4, 5, 5, 10 };

**int** ar31[] = { 5, 5, 10, 20 };

// Output: 5, 5

list = icp.commonIn3Arrs(ar11, ar21, ar31);

Assert.*assertTrue*(list.get(0) == 5);

Assert.*assertTrue*(list.get(1) == 5);

}

/\*\* 25. Count 1’s in a sorted binary array \*\*/

@Test

**public** **void** countOneInBinaryArrTest() {

**int** arr[] = { 1, 1, 1, 1, 0, 0, 0 };

Assert.*assertTrue*(icp.countOneInBinaryArr(arr, 0, arr.length - 1) == 4);

}

/\*\*

\* 26. Given a sorted array and a number x, find the pair in array whose sum is

\* closest to x

\*\*/

@Test

**public** **void** sumCLogestToXTest() {

**int** arr[] = { 10, 22, 28, 29, 30, 40 };

**int** x = 54;

List<Integer> list = icp.sumCLogestToX(arr, x);

Assert.*assertTrue*(list.get(0) == 22);

Assert.*assertTrue*(list.get(1) == 30);

}

/\*\* 27. Find the closest pair from two sorted arrays \*\*/

@Test

**public** **void** sumCLogestToX2ArrTest() {

**int** ar1[] = { 1, 4, 5, 7 };

**int** ar2[] = { 10, 20, 30, 40 };

**int** x = 38;

List<Integer> list = icp.sumCLogestToX(ar1, ar2, x);

Assert.*assertTrue*(list.get(0) == 7);

Assert.*assertTrue*(list.get(1) == 30);

}

/\*\* 28. K’th Smallest/Largest Element in Unsorted Array | Set 1 \*\*/

/\*\*

\* 29. K’th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear

\* Time)

\*\*/

/\*\*

\* 30. K’th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case

\* Linear Time)

\*\*/

/\* all above are common for minimum complxity O(k\*n) \*/

/\* @see @IOrderStatistics.java.. 1....program \*/

/\*\* 31. Find position of an element in a sorted array of infinite numbers \*\*/

@Test // @Amazn

**public** **void** posInInfiniteLenArrTest() {

**int** a[] = { 1, 2, 3, 4, 5, 6, 7, 8 };

**int** pos = icp.posInInfiniteLenArr(a, 5, 1000, Integer.***MAX\_VALUE***);

Assert.*assertTrue*(pos == 4);

}

/\*\*

\* 32. Given a sorted and rotated array, find if there is a pair with a given

\* sum

\*\*/

/\* in single array \*/

@Test

**public** **void** sumEqualToXTest() {

**int** a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

List<Integer> list = icp.sumEqualToX(a, 12);

Assert.*assertTrue*(list.get(0) == 9 || list.get(0) == 3);

Assert.*assertTrue*(list.get(1) == 9 || list.get(1) == 3);

}

/\* it two array \*/

@Test

**public** **void** sumEqualToX2ArrTest() {

**int** a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

**int** b[] = { 10, 20, 30, 40, 50, 60, 70, 80, 90 };

List<Integer> list = icp.sumEqualToX(a, b, 95);

Assert.*assertTrue*(list.get(0) == 5 || list.get(0) == 90);

Assert.*assertTrue*(list.get(1) == 5 || list.get(1) == 90);

}

/\*\* 33. Find the largest pair sum in an unsorted array \*\*/

@Test

**public** **void** largestPairTest() {

**int** a[] = { 1, 20, 3, 40, 5, 60, 7, 80, 9 };

List<Integer> list = icp.largestPair(a);

Assert.*assertTrue*(list.get(0) == 80);

Assert.*assertTrue*(list.get(1) == 60);

}

/\*\* 34. Find the nearest smaller numbers on left side in an array \*\*/

@Test

**public** **void** nearestSmallrOnLeftSideArrTest() {

**int** a[] = { 1, 6, 4, 10, 2, 5 };

**int** rs[] = { 0, 1, 1, 4, 1, 2 };

a = icp.nearestSmallrOnLeftSideArr(a);

**for** (**int** i = 0; i < a.length; i++)

Assert.*assertTrue*(a[i] == rs[i]);

}

/\*\* 35. K’th largest element in a stream \*\*/

/\* @see @IOrderStatistics.java....3....program8 \*/

/\*\* 36. Find a pair with maximum product in array of Integers \*\*/

/\* this is same max sum pair just use the product without absloute \*/

/\*\* 37. Find the element that appears once in a sorted array O(logn) \*\*/

@Test

**public** **void** uniqueElementTest() {

**int** arr[] = { 1, 1, 2, 4, 4, 5, 5, 6, 6 };

// System.out.println(icp.uniqueElement(arr, 0, arr.length - 1));

Assert.*assertTrue*(icp.uniqueElement(arr, 0, arr.length - 1) == 2);

}

/\*\* 38. Find the odd appearing element in O(Log n) time non sorted \*\*/

/\* logn in unsorted array \*/

@Test

**public** **void** searchIndexTest() {

**int** a[] = { 1, 9, 2, 8, 3, 7, 4, 6, 5 };

**int** n = a.length - 1;

Assert.*assertTrue*(icp.searchIndex(a, 0, n, 3) == 4);

Assert.*assertTrue*(icp.searchIndex(a, 0, n, 4) == 6);

Assert.*assertTrue*(icp.searchIndex(a, 0, n, 5) == 8);

}

@Test

**public** **void** oddAppearingTest() {

**int** arr[] = { 1, 1, 2, 2, 1, 1, 2, 2, 13, 1, 1, 40, 40 };

Assert.*assertTrue*(icp.uniqueElement(arr, 0, arr.length - 1) == 13);

}

/\*\* 39. Find the largest three elements in an array O(n) \*\*/

@Test

**public** **void** largestThreeElementTest() {

**int** arr[] = { 12, 13, 1, 10, 34, 1 };

**int** b[] = { 12, 13, 34 };

arr = icp.largestThreeElement(arr);

// System.out.println(Arrays.toString(arr));

**for** (**int** i = 0; i < b.length; i++)

Assert.*assertTrue*(arr[i] == b[i]);

}